Redpaper

Pierre-André Paumelle

Proven practices for enhancing performance: A Q&A for IBM WebSphere ILOG JRules 7.0.x

Many different aspects should be considered when addressing performance questions for a business rule management system (BRMS). Historically, many of the questions have centered around rule execution, such as “How fast is the rule engine?”, “How much memory does it require?”, and “How many CPUs will I need to execute 1,000 rules per second?” As illustrated in this document, the answers do not always come from the engine’s intrinsic capabilities, but depend upon the application and the way the Rule Execution Server is used.

As rule projects grow, there are also more and more performance topics related to other modules, such as build time in Rule Studio, memory and CPU usage in Rule Team Server and recommendations around Decision Validation Services usage.

The BRMS Service is realized through a series of products that can be installed:

- http://www.ibm.com/software/integration/business-rule-management/jrules/
- http://www.ibm.com/software/integration/business-rule-management/rule-team-server/
- http://www.ibm.com/software/integration/business-rule-management/decision-validation-services/

IBM® WebSphere® ILOG JRules 7.0 comprises a set of modules that operate in different environments, but also work together to provide a comprehensive BRMS. Figure 1 on page 2 shows the different modules in the environments in which they are used, and how they work together through synchronization.

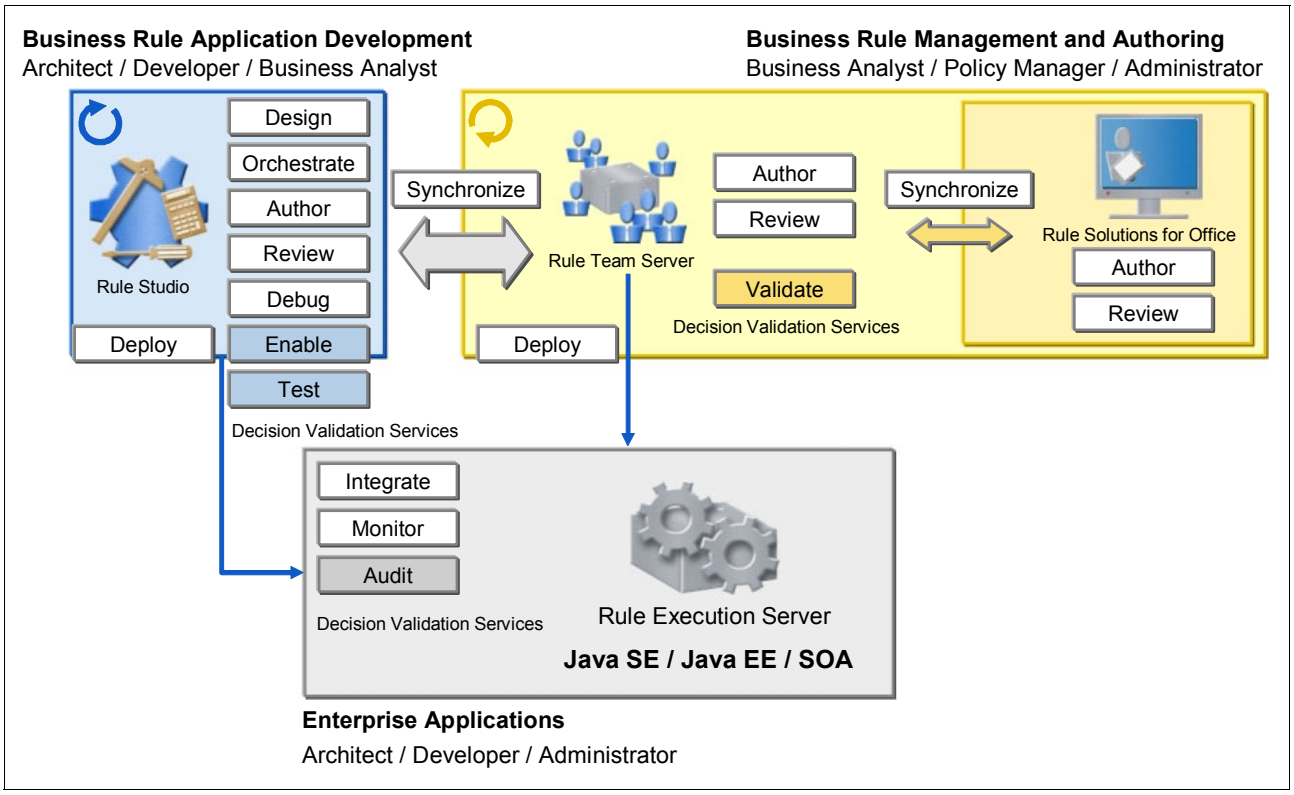

Figure 1 Product architecture

The purpose of this IBM Redpaper™ is to provide information about the performance aspects of the WebSphere ILOG BRMS 7.0.x. A question and answer (Q&A) format is used to cover as many dimensions as possible. Furthermore, performance is considered from various perspectives in this document, with an emphasis on production environments and execution.

As Rule Execution Server plays a key role in performance, this document starts by answering questions about this module.

The glossary of BRMS terms used in this document could be found in the WebSphere ILOG JRules glossary at the following Web page:

`http://publib.boulder.ibm.com/infocenter/brjrules/v7r0/topic/ilog.rules.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global2027.html`

Enhancing the performance of Rule Execution Server

The Rule Execution Server (Figure 2) is a complete execution and management environment for business rule applications. It provides a number of features that are required for high-performance, scalable and manageable applications:

- Pooling of rulesets and contexts
- Hot deployment of rulesets and asynchronous parsing
- File and database persistence of deployed rulesets
- Web-based system administration console
- Runtime monitoring of execution using the Java™ Management Extensions (JMX) API
- Client APIs: stateless Plain Old Java Object (POJO), stateful POJO, stateless EJB, stateful EJB, stateless JSE, stateful JSE, asynchronous execution and Message Driven EJB for asynchronous invocation, with optional execution trace or with optional auditable execution trace recording (Decision Warehouse).
- Easy service-oriented architecture (SOA) deployment: One-click configuration and integrated monitoring of rules as Transparent Decision Services
- Provision for simple testing and simulation of effects of rule changes (Decision Validation Services).

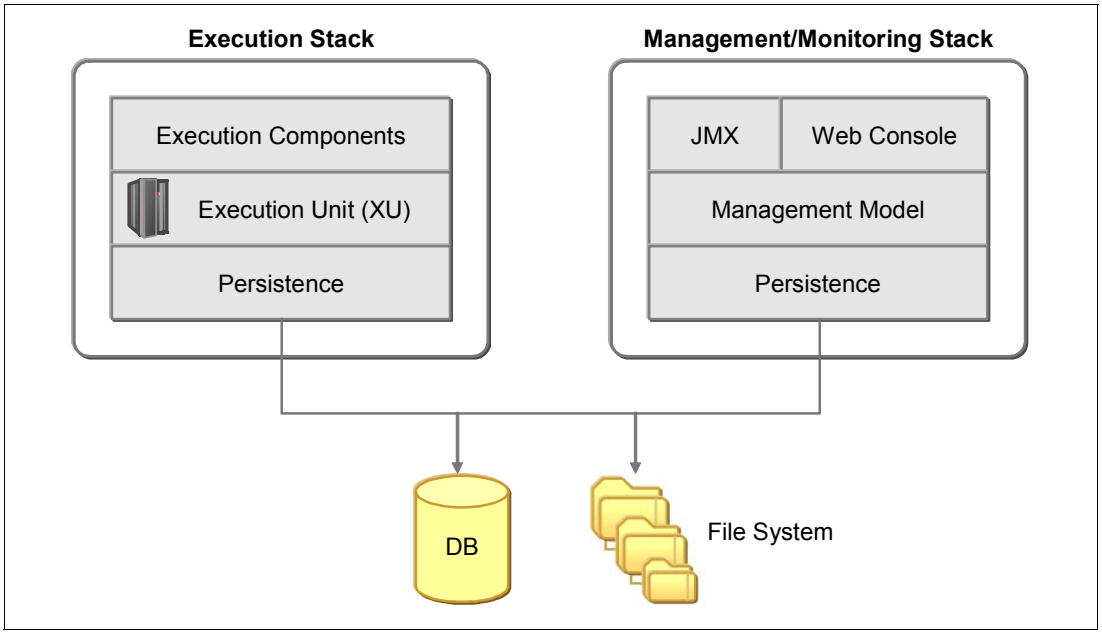

Figure 2 Product architecture

By deploying the Rule Execution Server (for JEE or JSE applications), your business rule applications automatically take advantage of the execution and management services offered by JRules, and you avoid having to implement ruleset and context pooling within your application code. Your overall application build and deployment processes can also integrate the Ant tasks IBM provides for the Rule Execution Server.

What is the basic configuration for good performance?

Use the following guidelines for the basic configuration settings:

- The log level in the Rule Execution Server should be set to level Severe or Warning in the production environment to increase performance. This property is accessible in the resource adaptor of the Rule Execution Server or in the `ra.xml`.
- The Execution Unit (XU) pool size should be tuned to have enough connection but with a reasonable memory consumption.
- The ruleSession factory should be unique and shared between HTTP sessions, otherwise a lookup JNDI is performed for each factory creation.
- If you do not use stateful rule sessions and the transactions at engine level, you should deactivate the transaction support in the Rule Execution Server and set NoTransaction instead of LocalTransaction in the properties of Rule Execution Server (XU) or in the `ra.xml`. If you have some code in your Executable Object Model (XOM) or rules that use a data source, the call to the data source can participate in the TX even if the XU transaction support is disabled. The XU transaction support set in the `ra.xml` only concerns the engine runtime itself and not other resources, even if called by the engine.
- Use up-to-date ruleSessions libraries:
  - Do not use old versions.
  - Integrate in the classpath of your application the correct version of the ruleSession.
- Tune the GC and memory size. On IBM JVM, the gencon GC policy has good results on small and medium applications.
- Use the asynchronous parsing of the Rule Execution Server if you have ruleset update in the application's use cases. This option is available in the properties of Rule Execution Server (XU) or in the `ra.xml`.
- Use the execution trace and the Decision Warehouse only if it is mandatory, and tune the usage if needed (filtering)
- Limit the size of your XOM to useful classes
- When on, Java2security halves performance. Java2security should be off if possible especially on JEE.

How do ruleset characteristics impact performance?

The size of a ruleset has a big impact on:

- Parsing time
- Execution with traces (number of rules, tasks, ruleflow)
- Execution with Decision Warehouse (number of rules, tasks)

Performance is greatly impacted by the following issues:

- Ruleflow complexity (Note that the ruleflow is interpreted, not compiled, in JRules 7.0.).
- The usage of dynamic filter in the rule tasks, use fixed list of rules if you can.
- Decision table (DT) complexity: Verify the size of the DT in the Rule Studio technical view. Dividing a DT several times can sometimes have a huge impact on the ruleset size by reducing it.
- Algorithm selection (see "How can we decide on Rule Engine Execution modes? " on page 22.).

Hints

- Filtering what gets traced can help minimize the performance overhead associated with execution tracing and the decision warehouse.
- A ruleset with mixed algorithms (RETE + Sequential or RETE + Fastpath) consumes more memory than a ruleset with a single algorithm.
- You can use the ruleflow with the mutual exclusive rule tasks to reduce the number of rules evaluated during the execution. In this way, the ruleset is segmented and only rules from relevant ruleTasks are evaluated.

How does a ruleset's Executable Object Model (XOM) impact performance?

XOM impacts performance in the following ways:

- Execution of JRules calls to XOM methods, so the performance of the XOM is crucial. The Java XOM must be thread-safe for the stability of the system.
- The Java XOM must implement Serializable if you call the Rule Execution Server remotely.
- The size of the Java XOM deployed in your EAR has an impact at the parsing and execution level. (Factor 3 or 5 in the pathologic cases). The class loader should be the smallest possible. For example: Do not put the complete JEE jars in the WEB_INF/lib directory of your Web application.
- A ruleset on an XML XOM should be configured to run in multiple simultaneous executions by configuring the pool of XML document drivers. The ruleset property ruleset.xmlDocumentDriverPool.maxSize configures it. The default value is one.
- The toString() method performance could have a huge impact if you ask an execution trace with the trace filter setInfoBoundObjectByRule sets to true.

What impact does the execution mode have on performance? (Trace, Decision Warehouse)

Consider the following aspects of execution mode:

- No trace:
  - Execution without traces is the optimal mode in terms of memory usage and performance.
- Execution Trace:

  When you choose the execution mode with trace, you experience the following side effects:

  - Memory usage increases as the execution trace is generated, the list of rules and tasks is cached, and the rule events are generated.
  - On sequential or Fastpath, the ruleset property ruleset.sequential.trace.enabled set at true enables the generation of events when a rule is fired or a task executed. The side effect is a code generation that produces event and store mapping information in the ruleset (performance and memory impact).
  - The response of the execution is bigger than the original one as the execution trace is integrated. The execution trace size depends on the ruleset characteristics (number of rules and tasks).
  - The trace filtering could have a big impact on the size of the execution trace and, therefore, performance.

- Decision Warehouse execution is divided into the following steps:
  - This execution mode depends on the execution trace mode, so we pay for the cost of the execution trace.
  - At the end of execution, a serialization of input and output parameters is done (BOM serialization).
  - A complete serialization to XML of the execution trace.
  - A database insertion.
  - You can choose through a list of filters (as ruleset properties) what should be saved in the Decision Warehouse.

Note: In the current benchmarks for the Decision Warehouse, the key bottleneck is the database insertion. This makes database tuning important if you use the Decision Warehouse out of the box.

What impact does the topology have on performance? (JSE embedded, JEE shared, JEE isolated)

The rule execution server can be deployed on three different topologies. Consider the following information when evaluating the impact of the topology on performance.

- JEE Shared:
  - The XU is deployed to the application server and shared (and possibly accessed) by all the applications deployed to the server.

    This is analogous to installing a device driver into an operating system, in that the application server has become globally enhanced with ruleset execution capabilities.
  - From an administrative perspective, one version of the XU is deployed and can be easily upgraded, started, stopped and monitored using the application server management console or other tools.
  - The XU should be deployed on all nodes of the cluster.
  - The JEE ruleSessions types are available.
- JEE Isolated:
  - The XU is deployed within a single application.
  - This is a more advanced deployment option that enables multiple applications to be deployed to the same server, and for each application to use completely separate versions of ILOG JRules classes.
  - The XU Jrules-res-xu-xxx.rar is embedded in the customer application
  - The resource is scoped to the customer application.
  - The clustering is linked to the customer application.
  - The JEE ruleSessions types are available.

- JSE Embedded:
  - The XU is deployed as a simple Java Standard Edition JAR and Rules implements a lightweight J2C container, as well as the Resource Adapter.
  - This mode allows the XU (and hence, the Rule Execution Server) to be used outside a JEE application server.
  - The application server no longer manages the Resource Adapter in this mode and the ILOG JRules pooling infrastructure is used.
  - The resource is JSE and scoped to customer application.
  - Only the JSE ruleSession is available.
  - Performance impacts:
    - The performance of JEE shared and JEE isolated Rule Execution Server are equivalent.
    - The performance of JSE embedded are similar with JEE shared but the pool cannot be configured by the application server administrator

How can we optimize the XU pool size and minimize the parsing impact?

The optimal XU pool size depends on the number of possible concurrent connections and the number of rulesets to execute. The limit is reached by the memory consumption. On a dual-core computer, a pool size of 50 is correct.

The following three scenarios illustrate how the performance overhead of ruleset parsing can be minimized:

- Parsing at Rule Execution Server launch or first invocation of the ruleset. In this case, the ruleset should be parsed before the first execution. You can force it with a dummy execution or a call of the loadUptodateRuleset in IlrManagementSession API (Management Session API).
- Update of a ruleset already in use. This can be fixed by using the asynchronous parsing.
- A ruleset is used infrequently once a day, for example, and removed from the pool running other rulesets. This can be fixed by opening a stateful rule session. In this case, the ruleset will stay in the pool when your session is open.

See the White Paper *Minimizing the effect of ruleset parsing on the performance of Rule Execution Server* for details. You can access the White paper at the following Web page:

`ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/wsw14089usen/WSW14089USEN.PDF`

Rule Execution Server and Clustering?

The Rule Execution Server is ready for clustering. Each XU is independent at pool level but shares the same Rule Execution Server persistence. The XUs do not share a parsed ruleset, so each time a ruleset is run for the first time or if the ruleset is not in the pool on a node, the XU will parse the ruleset.

Clusterwide JMX Servers (as provided by most application servers) allow the Rule Execution Server Management Console to discover and interact with multiple XUs no matter where or how they are deployed across the cluster. Providing a single administration facility for all the rule engines deployed across a cluster of machines allows system administrators to gain an understanding of the deployment topology and diagnose runtime or configuration errors.

What is the tooling for analyzing the status of the Rule Execution Server?

The Rule Execution Server's status is visible through a tool accessible through the server's console.

Retrieving ruleset caching details from Rule Execution Server Console

You can get detailed ruleset caching information (XU dump) from the Rule Execution Server Console. This includes how many free connections there are in the pool, which rulesets have been parsed, how much free memory is left, and so on.

To access ruleset caching information:

1. Change the Log Level of the Rule Execution Server Console to Debug (Figure 3).

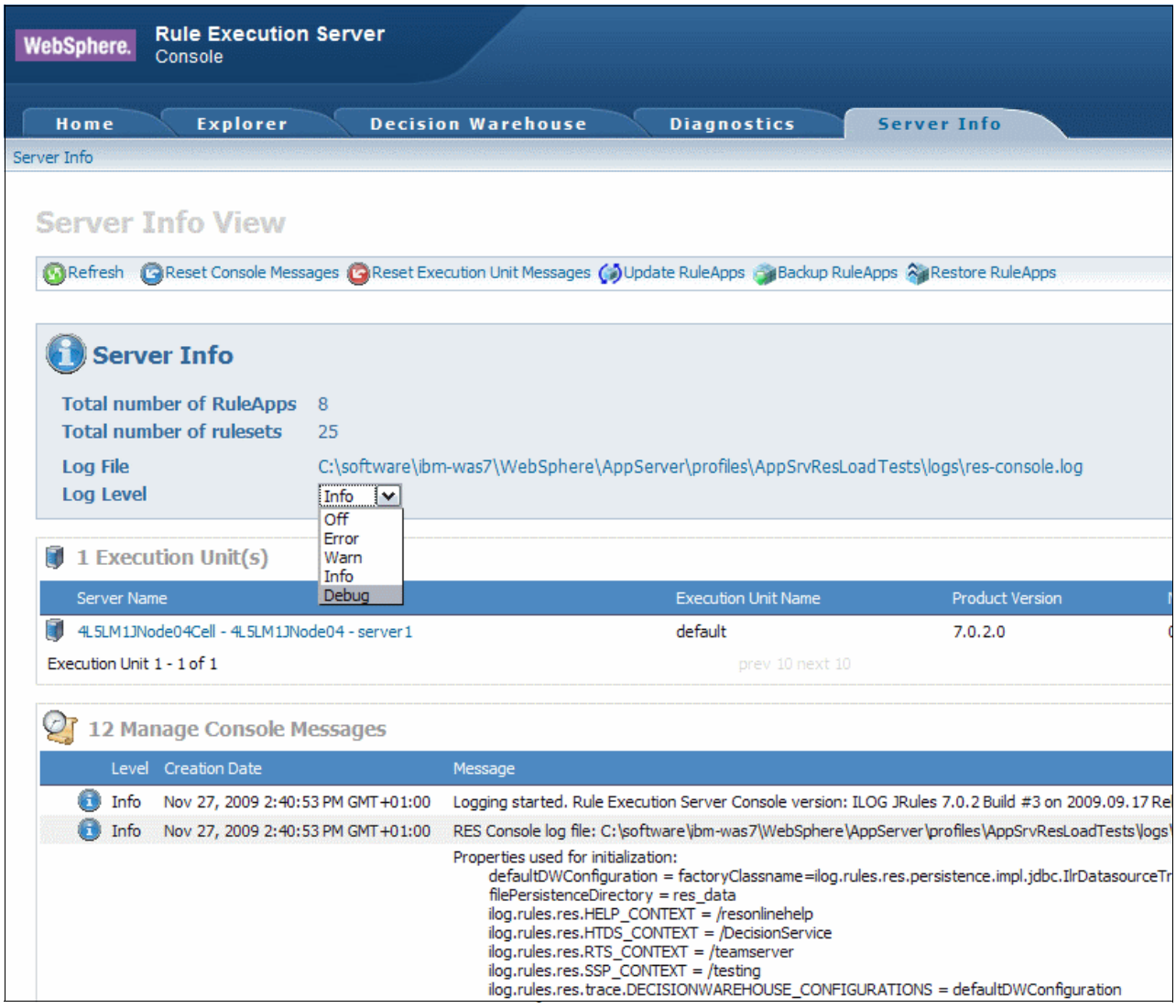

Figure 3 Status of the Rule Execution Server

2. Navigate to the XU you want to investigate (Figure 4).

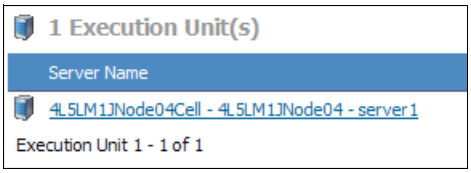

Figure 4 Status of the Rule Execution Server

3. Click View (Figure 5).

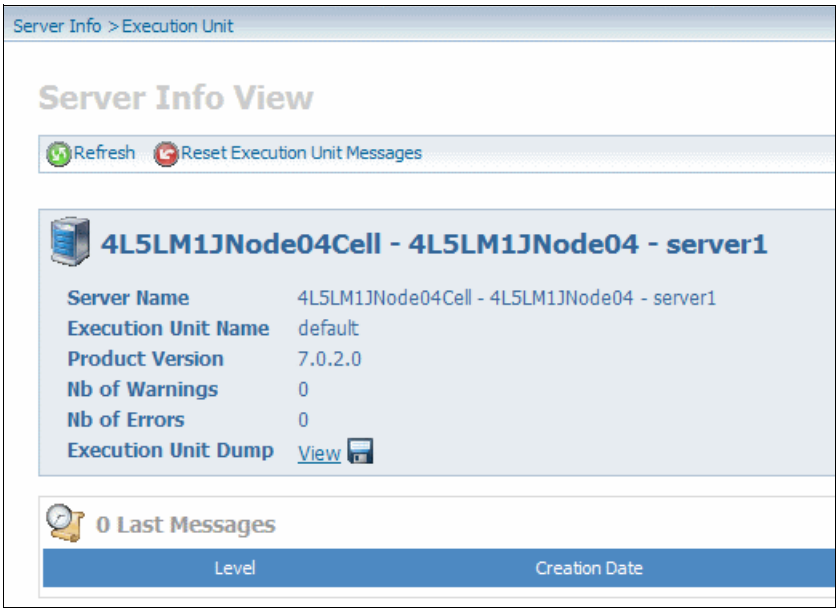

Figure 5 Status of the Rule Execution Server

You can see which rulesets have been parsed and are stored in the cache. Figure 6 shows that two rulesets are currently stored in the cache.

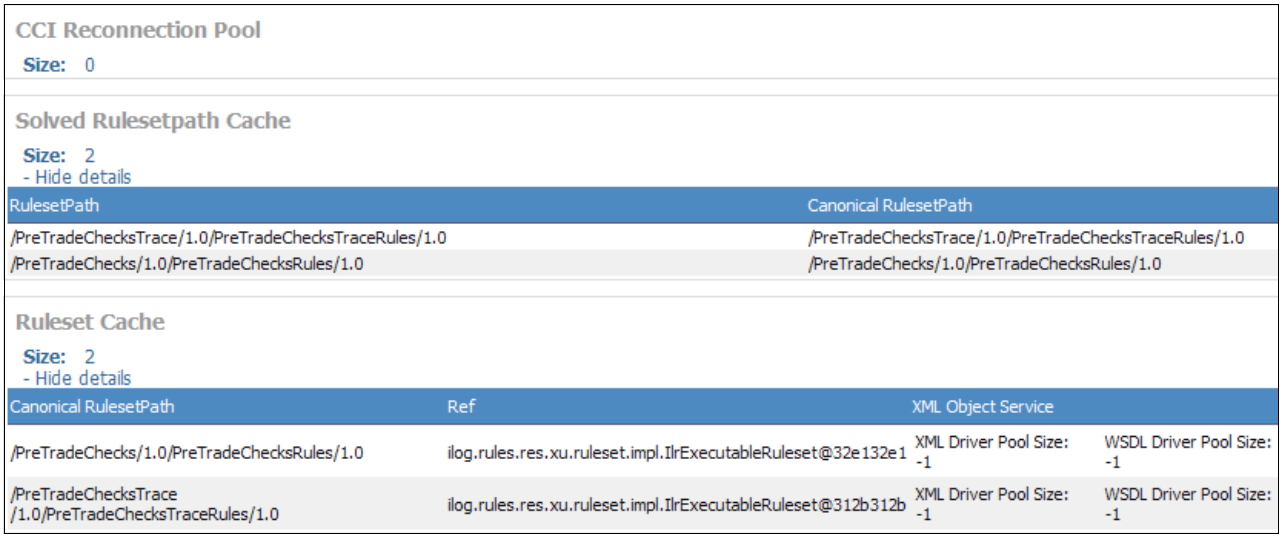

Figure 6 Status of the Rule Execution Server

Do you have benchmarks on Rule Execution Server?

We provide the following benchmark to show the impact that the number of rules can have on performance without modifying the semantic of the original ruleset. The number of rules fired is always the same, and the result is always the same. The only difference is the number of rules. The benchmark is performed with IBM Rational® Performance Tester 8.1, which created 35 virtual users calling the same Web application, to drive the ruleset through a POJO rule session. All benchmarks ran on IBM WebSphere Application Server 7, with IBM WebSphere ILOG JRules 7.0.1 installed with a shared instance of Rule Execution Server (for example, JEE shared topology). The duration of each bench was one hour to reduce the impact of first execution.

The benchmarks configuration spanned three machines on the same subnet:

- One machine with the Rational Performance Tester agent
- One machine with IBM WebSphere Application Server V7, the Rule Execution Server and the benchmark application
- One machine with a database

You can find useful information about Rational Performance Tester at the following Web pages:

- Rational Performance Tester site

  http://www.ibm.com/developerworks/rational/products/performancetester/
- Rational Performance Testing Forum

  http://www.ibm.com/developerworks/forums/forum.jspa?forumID=326

The software and hardware used was:

- zLinux
  - OS z/Linux® (SUSE® on IBM OS/390®. Kernel = 2.6.16.60-0.42.5-default)
  - Memory 2 gigabytes
  - CPU 3 processors
  - WAS 7.0.0.3 (base)
  - 31 bits JVM
- Distributed
  - OS Windows® XP SP3
  - Memory 2 gigabytes
  - CPU 2 Xeon 2.8 GHz
  - WAS 7.0.0.3 (base)

As you can see in Figure 7, the number of rules does not have a direct impact on the execution of rulesets (measured as transactions per second) with the same semantic.

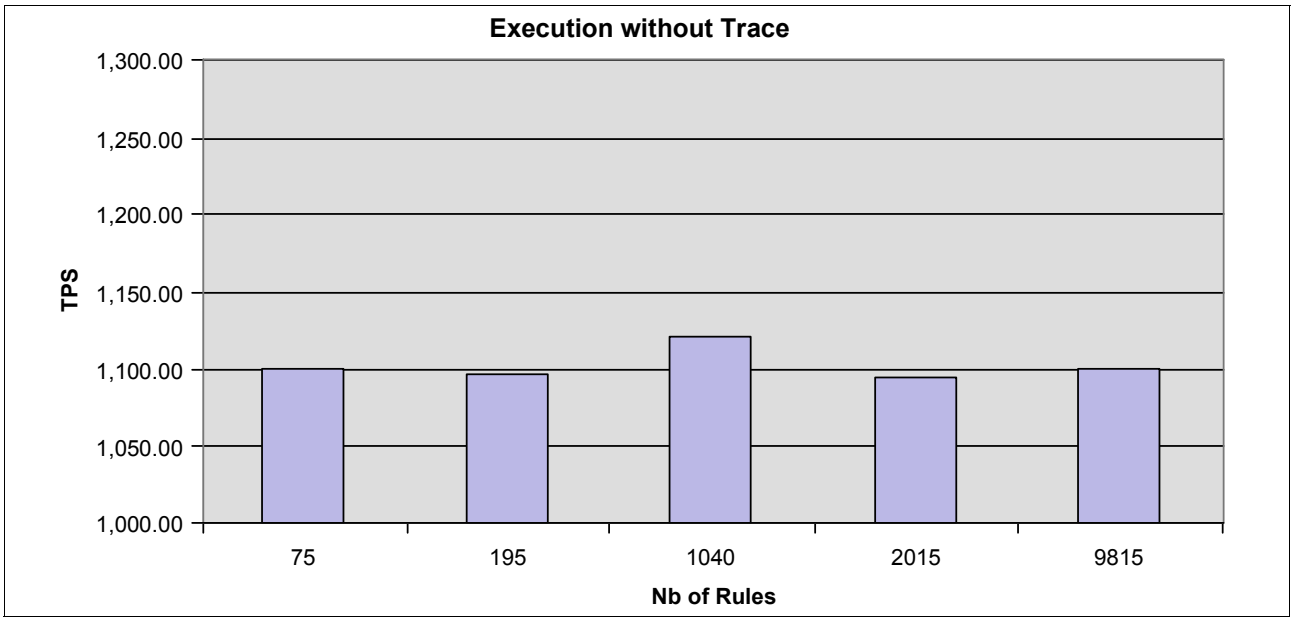

Figure 7 Benchmarks on zLinux

Figure 8 shows that the number of the rules has a direct impact on the execution with default traces. These traces contain the complete list of rules so it is not a surprise.

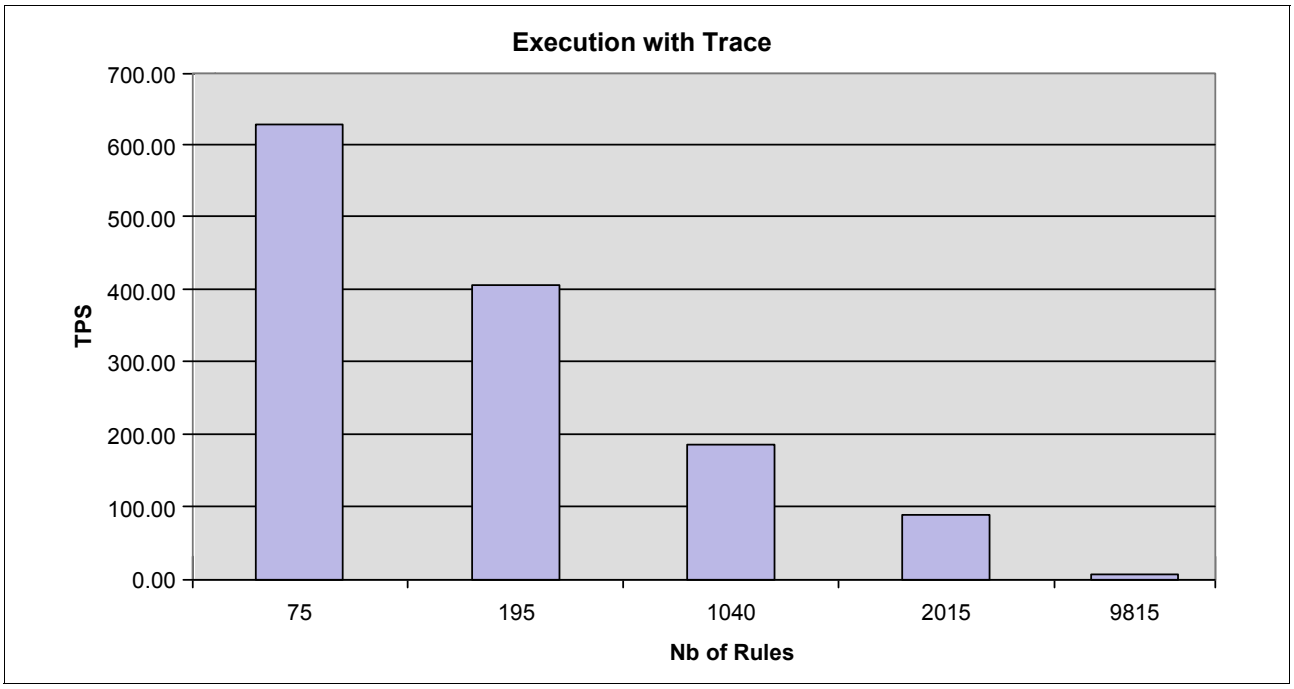

Figure 8 Benchmarks on zLinux

Figure 9 shows that the number of rules has a direct impact on the ruleset parsing duration (measured in milliseconds). This time is close to the response time for the first execution.

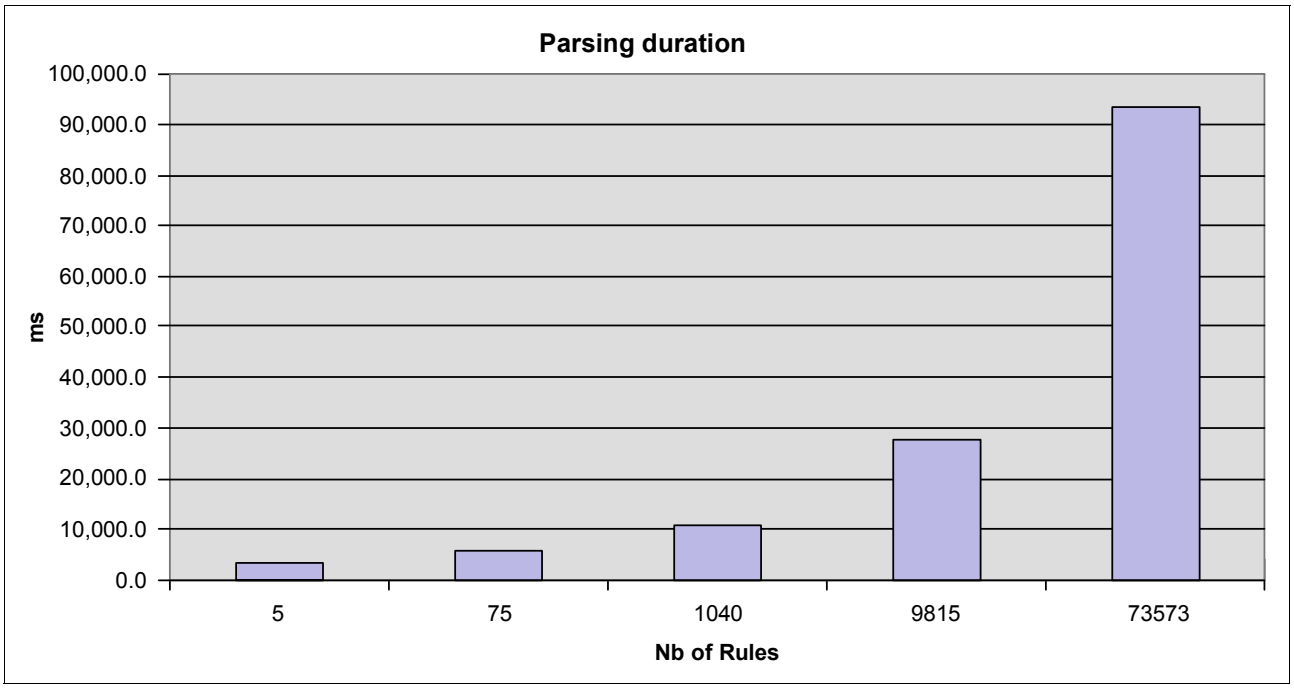

Figure 9 Benchmarks on distributed

Enhancing the performance of Decision Warehouse

The Decision Warehouse traces decisions in production applications. Figure 10 shows the interface used to search and view decisions stored in the warehouse.

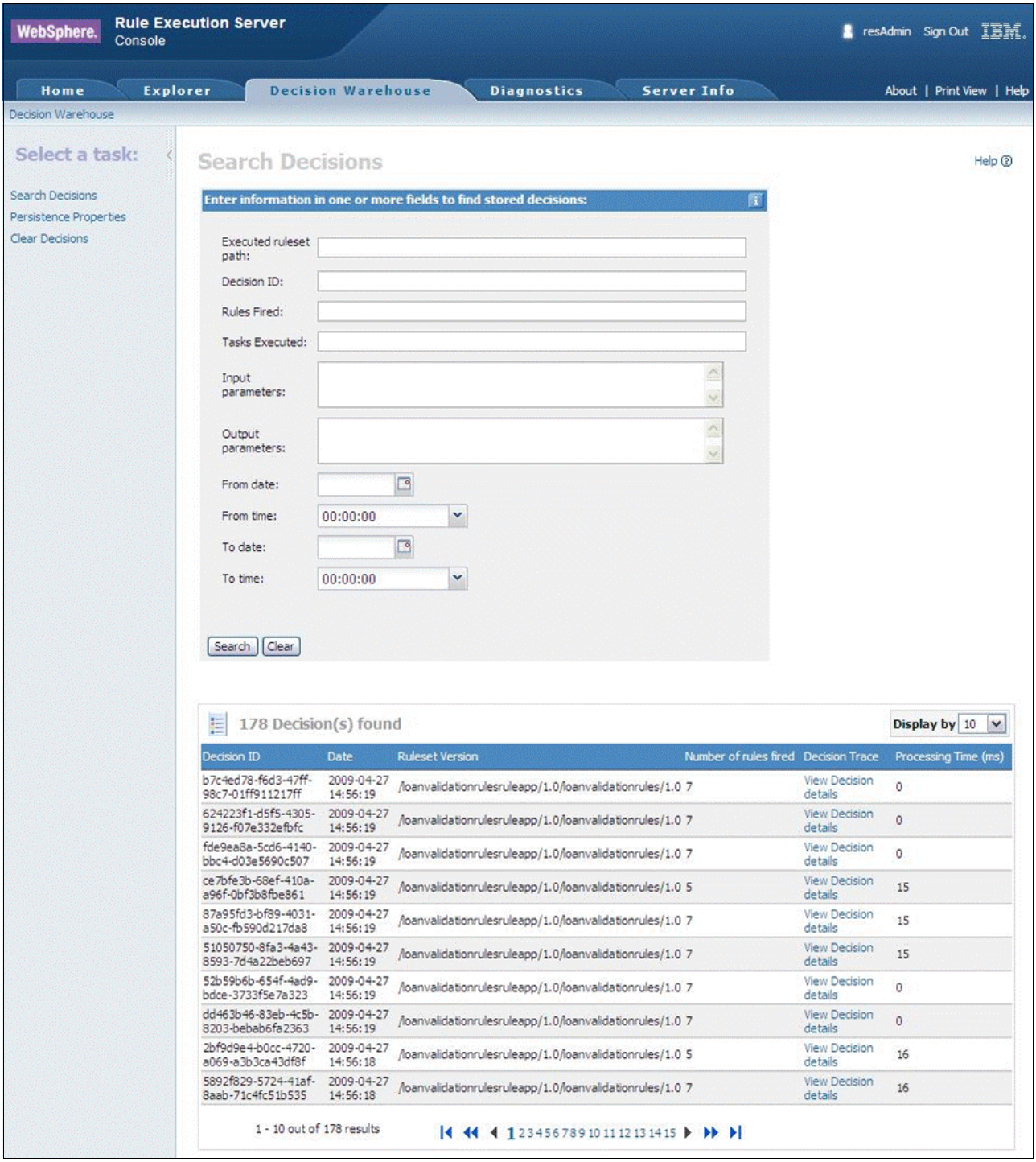

Figure 10 Decision Warehouse

The features of Decision Warehouse are:

- Logs execution trace
- Input / Output data
- Execution results
- Executed tasks
- Rules fired
- Queries
- Open API to connect third-party BI tools

Figure 11 shows an overview of the Decision Warehouse function.

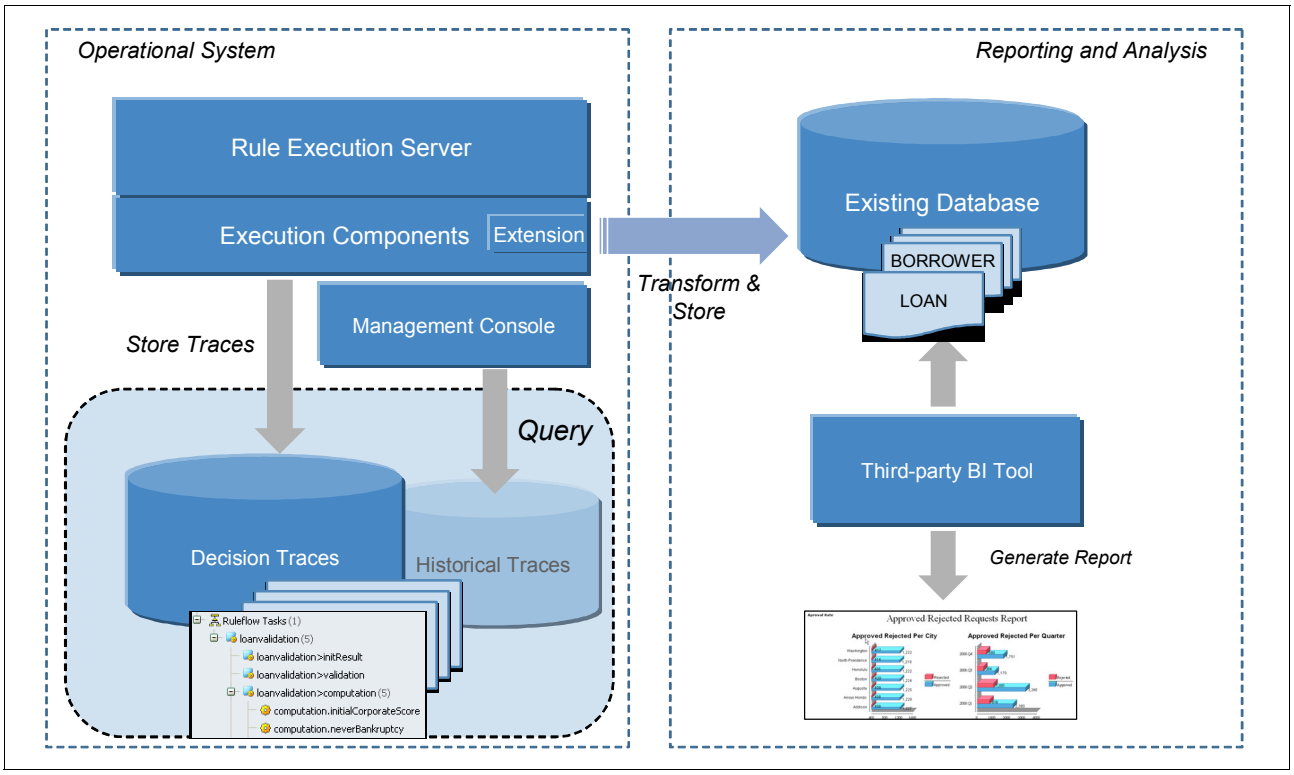

Figure 11 Decision Warehouse

What impact does the Decision Warehouse have on the Rule Execution?

The performance of an application greatly depends on the following characteristics:

- The complexity of the object model

  If the ruleset signature contains input/output parameters that have a large amount of data, it can take a long time to serialize and persist.

- The number of rules within the ruleset

  Performance degrades as the number of rules fired increases.

- The level of detail you want to capture

  The more execution details you want to capture, the longer it takes to complete the execution.

In general, the performance of the Decision Warehouse depends on:

- The execution trace generation performance
- The serialization performance, which divides into two principal cases for the input/output parameter serialization to BOM (depends on the size of the ruleset parameters) and the serialization of the execution trace, which depends on the ruleset characteristics (number of rules, number of rules fired …)
- The performance of the database server and the network used to access it.

How can we optimize the Decision Warehouse?

There are three main ways to optimize the Decision Warehouse:

- The Decision Warehouse is configurable at a ruleset base with filter properties. With this, you can choose the information to save. For example, there are configurable parameters to specify the ruleset properties to optimize performance. You can choose which execution information you want to capture from this list:
  - Executed ruleset path
  - Decision ID
  - Rules fired
  - Tasks executed
  - Input/output parameters
  - Execution duration
  - Tasks not executed
  - Rules not fired
  - All the names of the rules/tasks
  - Number of rules/tasks executed
  - Custom execution output
- The performance of the Decision Warehouse depends on the database. For example, the database should be optimized to handle CLOB data, not in a generic way, but for the specific case of the ruleset.
- The Decision Warehouse is customizable. Further customization can be done through the Decision Warehouse extension point for better execution performance. You can define how data can be persisted or queried through the extension point. See *Decision Warehouse customization sample* at:

  http://publib.boulder.ibm.com/infocenter/brjrules/v7r0/topic/ilog.rules.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global1710.html

  You can also implement your own asynchronous logging of the data captured with the Decision Warehouse feature. There is a sample in the product, which shows the reference architecture for using the extension point.

There is a section in the documentation that describes the ways to optimize execution performance when using Decision Warehouse. See *Optimizing Decision Warehouse* at the following Web page:

http://publib.boulder.ibm.com/infocenter/brjrules/v7r0/index.jsp?topic=/ilog.rules.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global758.html

Enhancing the performance of Rule Team Server

Designed to enhance productivity, WebSphere ILOG Rule Team Server (Figure 12) brings unsurpassed levels of usability and sophistication to business rule management. Distributed business teams collaborate with a powerful, easy-to-use Web environment to create, manage, and deploy business rules.

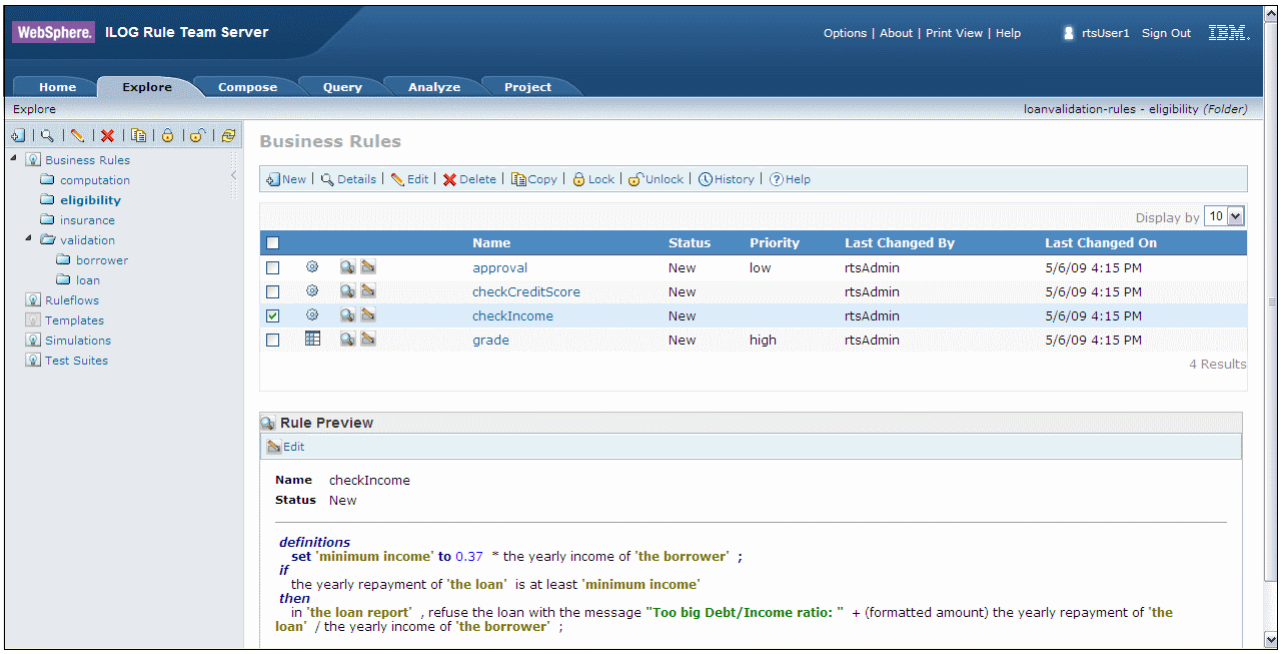

Figure 12 Rule Team Server

Manage rules across the enterprise with confidence

Rule Team Server is easy to learn, use, and deploy. Business users can manage rules quickly and securely, saving time and money. Audit reporting and access to a central rule repository give users the peace of mind they need to focus on other tasks, keeping them ahead in a competitive business environment.

With WebSphere ILOG Rule Team Server you can:

- Author and edit rules using a natural language and customizable business vocabulary. Rules can be expressed in graphical formats, such as decision tables and decision trees.
- Simplify rule projects through Smart Views (user configurable navigation panes), filters, and reports.
- Manage collaboration with role-based permissions, access controls, and customizable rule metadata properties.
- Facilitate rule maintenance with templates, point-and-click editors, error checking, and versioning.
- Ensure rule quality with customizable queries, rule analysis, and visual comparisons of changes between versions.
- Add powerful testing and simulation capabilities using Rule Team Server's integration with WebSphere ILOG Decision Validation Services.
- Extend rule management across the enterprise through Rule Team Server's integration with WebSphere ILOG Rule Solutions for Office.

General guidelines and considerations on Rule Team Server

Rule Team Server makes heavy use of CPU, memory, and database resources. To ensure scalability, the hardware architecture should be set up to support usage requirements. Although it is difficult to give precise recommendations due to the variety of possible configurations, this section should help you understand where possible bottlenecks can occur.

What is the memory footprint of Rule Team Server?

Rule Team Server is designed to be accessed by several users at the same time. Each time a user accesses the Rule Team Server URL, an HTTP session is created on the server and filled with several objects, including the current IlrSession to connect to Rule Team Server, some Java Beans to support the model of the JSF components used in the interface, and some Web components (guided editor and decision table editor). The memory footprint required for a session is difficult to estimate because it depends upon many factors, including what actions are performed through the GUI, the Virtual Machine (VM), and the structure of the repository.

However, we can provide a specific example. After browsing a repository with 10,000 rules, expanding and collapsing nodes in the rule explorer, previewing some rules, editing one rule and generating a ruleset (with IRL already cached), the memory footprint of a session is around 10 MB.

During the ruleset generation, around 150 MB of short-lived objects are created. This can cause some issues if many users are performing this operation at the same time.

Hints

If you believe the number of users connected simultaneously to a single server will not fit into the memory allocated for a single VM, you should deploy Rule Team Server to a cluster of servers and use load-balancing, so that HTTP sessions are dispatched according to the server's available memory.

Another parameter to take into account when dealing with memory is the number and size of the vocabularies used in a Rule Team Server repository. The vocabulary of a given project is stored in memory the first time it is loaded and is shared across sessions. By default, vocabularies are stored in a pool of 10 instances. After 10 instances, other vocabularies are soft-referenced so that the VM can free the memory if needed. The same pooling algorithm applies to instances of BOMs and variable providers.

What is the guidance on CPU usage for Rule Team Server?

According to the number of users and which operations they frequently perform when using Rule Team Server, the number of CPUs can be critical. Parsing a BAL rule or a decision table is an atomic operation that takes a lock on the vocabulary instance to be used. In addition, as test results have revealed, most of the long-running operations (ruleset generation, reporting, and so forth) make heavy use of the CPU. This means that if many users are accessing the same project at the same time, browsing the repository, or running long operations such as ruleset generation, the shared vocabulary may become a bottleneck. This reinforces the need to deploy Rule Team Server on a cluster.

What are the hints for maintaining good database performance for Rule Team Server?

Most Rule Team Server operations perform SQL queries.

Hints

Since some queries can be time consuming, we recommend that you deploy the application server and the database on different machines to force the distribution of the load between Java computing and database access. This will improve the user experience when multiple users are connected to the repository.

We also recommend that you set the JDBC connection pool of the data source to a number that is equivalent to the maximum number of users that will simultaneously connect to Rule Team Server. If the pool is too small, users may end up getting connection timeouts when their sessions are trying to access the database.

Enhancing the performance of Decision Validation Service

Figure 13 shows an overview of the Decision Validation Services (DVS) architecture.

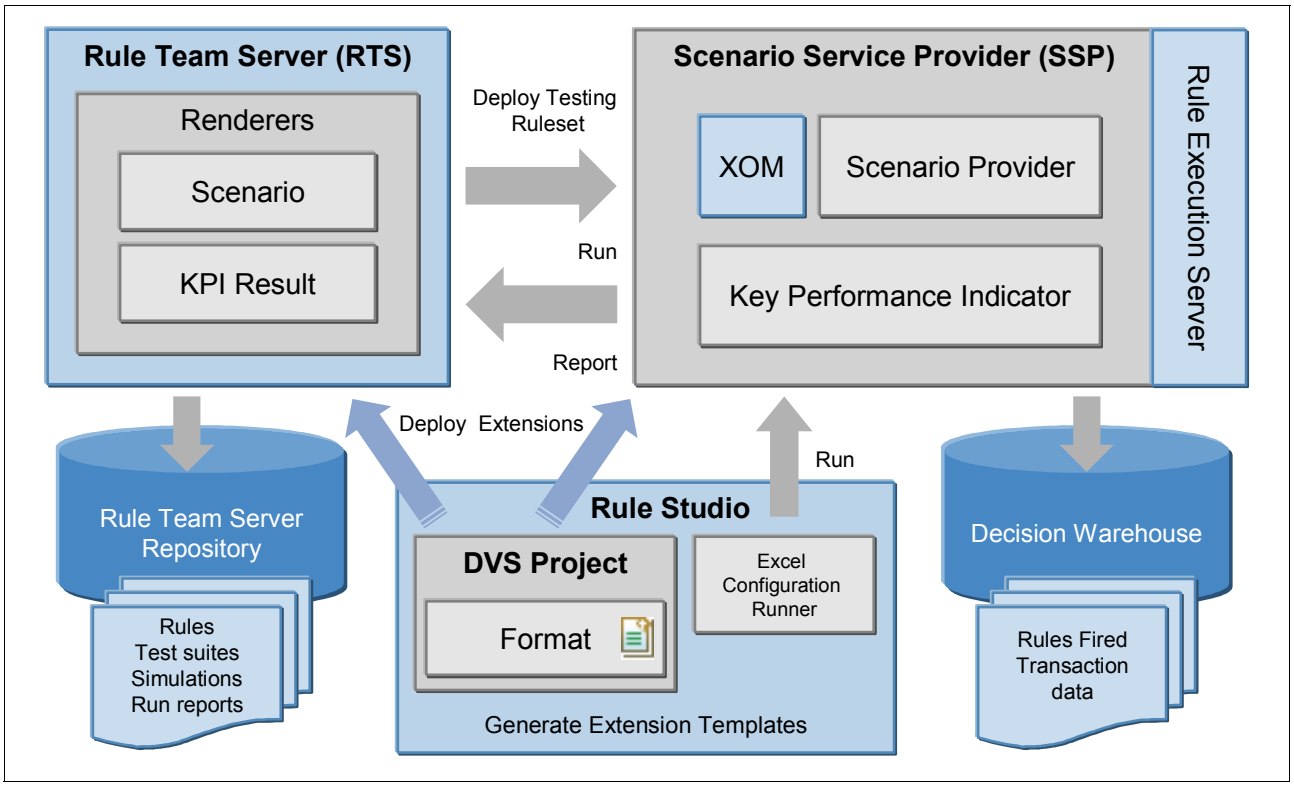

Figure 13 Decision Validation Services

The main features of the DVS are:

- Out-of-the-box ruleset testing (Rule Team Server)
- Business impact simulation (Rule Team Server)
- Scenario configuration and customization (Rule Studio)
- Audit - Decision Warehouse (Rule Execution Server)

What are some of the recommendations for DVS usage?

Executing a test suite or simulation can be a long operation, depending on the number of rules and the number of scenarios to execute. Here are the main steps performed during the operation:

1. Generate a ruleapp archive with the set of rules to test.
2. Deploy this ruleapp to a Rule Execution Server console.
3. Trigger the execution of the ruleapp on the Scenario Service Provider, which will pass the call to the XU.
4. Get the report.

Most of the time consumed on Rule Team Server will take place during ruleset generation. For a ruleset of 10000 business rules, if this is a second ruleset generation (for example, if the IRL is already cached), it will take 10 seconds (six minutes for first ruleset generation). The rest of the time (almost two minutes in our case) is taken in the Rule Execution Server (Scenario Service Provider/XU), parsing the generated archive and executing the scenarios.

Hints

We recommend running Rule Execution Server and Rule Team Server on two different machines, so that when Rule Execution Server consumes some resources executing a test suite or simulation, Rule Team Server resources (CPU and memory) are not impacted.

Rule Execution Server setting recommendations

Each time a test suite execution is triggered from Rule Team Server, a new ruleapp is created, deployed, and parsed on the Rule Execution Server. This means that the parsing cost on Rule Execution Server cannot be avoided in the context of DVS. From the tests performed, a 10,000-rule ruleset took almost all of the 768 MB of RAM allocated for the VM. This means that without any further configuration, a second execution launched just after the first one would require a new item in the XU pool, provoking an out of memory for the VM hosting the Rule Execution Server.

Therefore, we recommend the following:

- Dedicate a Rule Execution Server instance to DVS. Do not try to use a production Rule Execution Server for DVS tests or simulations.
- Set the size of the XU pool[1] according to the total available memory/maximum amount of memory of a ruleset. For example, if you have one GB allocated for your VM and are executing a ruleset that takes 300 MB of memory, the size of the pool for the XU must be lower than or equal to four. The size of the pool must not be under two, a test suite runs two rulesets the ruleset to test and the ruleset that contains the tests.
- Set the size of the Scenario Service Provider pool[2] to the same size of the XU pool. When this is done, the XU pool (and thus the memory of the VM) will not be overloaded uselessly, nor will there be a bottleneck for test suite/simulation executions.
- If you have a huge volume of scenarios, the report options usage has a great impact on performances as we use the Decision Warehouse to store the execution results.

[1] Set in ra.xml of the Scenario Service Provider, or configured in the JCA connection factory of the application server.
[2] JRULES_JOBS_POOL_SIZE in the web.xml of the Scenario Service Provider WAR file.

Enhancing the performance of Rule Engine Execution

The basic functions of the rule engine are:

- Read its rules dynamically at runtime.
- Reason on objects it knows.
- Keep track of changes to the objects it knows.
- Invoke the execution of rules, also called firing of rules.

An overview of the Rule Engine is shown in Figure 14.

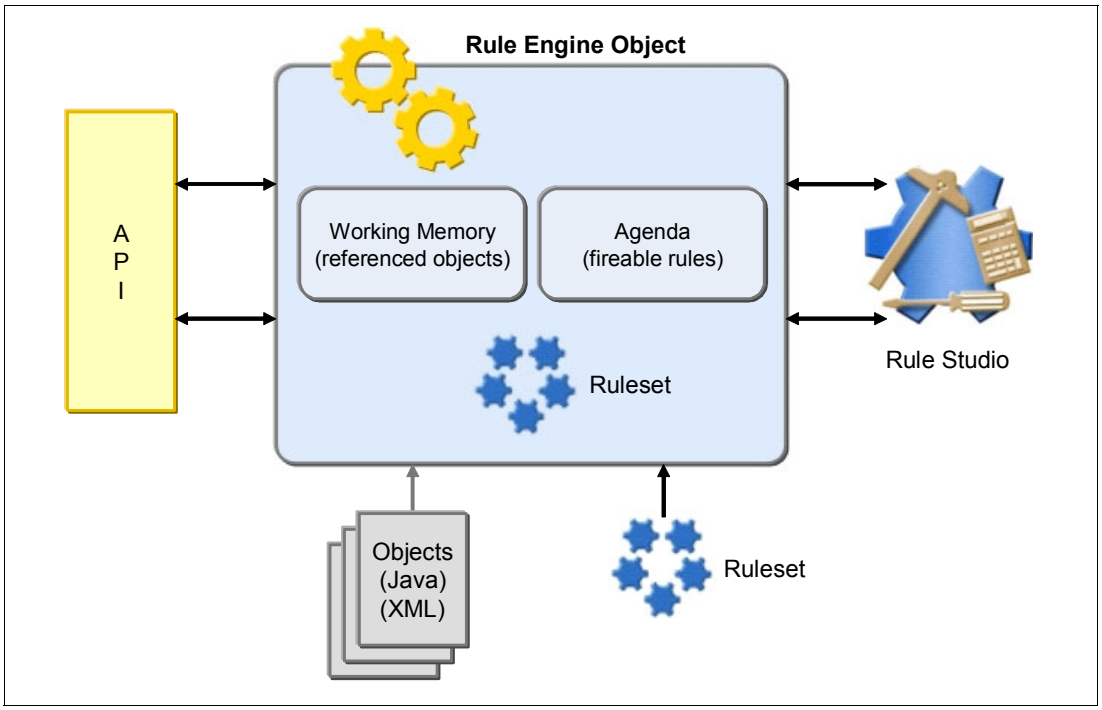

Figure 14 Rule Engine

The rule engine processes the rules provided to it. It evaluates the rules against the application objects and executes rule actions when appropriate. The rule engine can operate on ruleset parameters, local ruleset variables, and native objects inserted into the working memory (references). Rules are evaluated according to the ruleflows contained in the ruleset archive. If there is no ruleflow, all the rules in the ruleset are evaluated.

How relevant are academic benchmarks?

Here are some of the problems with academic benchmarks:

- Small number of rules (less than 50)
- Test worst-case RETE inference and agenda usage, which is rarely encountered with well-designed applications
- Implement solutions to combinatorial search problems, typically better implemented using a constraint programming (CP) engine rather than a RETE engine
- No usage of common patterns such as ruleflow and sequential execution
- The solutions are not verified, meaning that correctness may be sacrificed for speed. This is particularly the case for Waltz and WaltzDB.
- The benchmarks are easy to manipulate because every vendor ports from the original OPS5 source code into their own rule language, where they take advantage of their product's features.

For reference, here are descriptions of the three most famous academic benchmarks.

- Manners

  Manners is based on a depth-first search solution to the problem of finding an acceptable seating arrangement for guests at a dinner party. The seating protocol ensures that each guest is seated next to someone of the opposite sex who shares at least one hobby.

- Waltz

  Waltz was developed at Columbia University. It is an expert system designed to aid in the three-dimensional interpretation of a two-dimensional line drawing. It does so by labeling all lines in the scene based on constraint propagation. Only scenes containing junctions composed of two and three lines are permitted. The knowledge that Waltz uses is embedded in the rules. The constraint propagation consists of 17 rules that irrevocably assign labels to lines based on the labels that already exist. Additionally, there are four rules that establish the initial labels.

- Waltzdb

  Waltzdb was developed at the University of Texas at Austin. It is a more general version of the Waltz program. Waltzdb is designed so that it can be easily adapted to support junctions of four, five, or six lines. The method used in solving the labeling problem is a version of the algorithm described by Winston [Winston, 1984]. The key difference between the problem solving technique used in Waltz and Waltzdb is that Waltzdb uses a database of legal line labels that are applied to the junctions in a constrained manner. In Waltz, the constraints are enforced by constant tests within the rules. The input data for Waltz is a set of lines defined by Cartesian coordinate pairs.

Conclusions

If performance is one of your major buying criteria, you are encouraged to build a proof-of-concept (POC) set of rules and data and verify rule engine performance in your own environment. It is impossible to extrapolate from published academic benchmark results to your running application, with your rules and data, deployed to your OS and hardware. In addition, a POC will also allow you to evaluate the BRMS from as many angles as possible, spanning ease of use, business user accessibility, support and professional services, performance, deployment platforms, scalability, and so forth.

How can we decide on Rule Engine Execution modes?

You can choose an execution mode for each rule task in your ruleflow. By default, a rule task is assigned the RetePlus execution mode. To achieve optimal performance, you may need to choose another execution mode that is better adapted for the rules in a particular rule task.

Table 1 shows which mode to choose for each case:

Table 1 Quick view of execution mode selection

Choose this mode:	In this case:
RetePlus (the default mode)	Rule chaining; may be useful in the case of many objects
Sequential	Many rules, few objects; has limitations
Fastpath	Rules implementing a decision structure, many objects; may have longer compilation but faster at runtime.

To determine which execution mode you should use on a rule task, analyze the structure of the rules in the rule task, and what type of processing they perform. To make the best choice, you need to answer the following questions:

- What type of application do your rules implement?
- What types of objects are used by your rules?
- What is the impact of rule actions?
- What sort of tests do you find in rule conditions?
- What priorities have you set on your rules?

What type of application do your rules implement?

Depending upon the purpose of the business logic defined in a rule task, the execution mode you choose differs.

- Compliance and validation

  Loosely interrelated rules that check a set of conditions to yield a go/no-go or similar constrained result. Compliance business rule applications are often used in underwriting, fraud detection, data validation and form validation. The business rules in this kind of application would generally have a yes or no result, and provide some explanation on the decision.

  For this type of application, we recommend you use the sequential or Fastpath execution modes.

- Computation

  Strongly interrelated rules that compute metrics for a complex object model. Computational business rule applications are often used for scoring and rating, contracts and allocation. The business rules in this kind of application would carry out different calculations on an object, which would be responsible for providing a final value, or rating.

  For this type of application, we recommend you use the RetePlus execution mode.

- Correlation

  Strongly interrelated rules that correlate information from a set of objects to compute some complex metrics. Correlation business rule applications are often used for billing. The business rules in this kind of application insert information.

  For this type of application, we recommend you use the RetePlus or Fastpath execution mode.

- Stateful session

  Strongly interrelated rules that correlate events in a stateful engine session. Stateful applications are often used in alarm filtering and correlation, GUI customization, and Web page navigation.

  For this type of application, we recommend you use the RetePlus execution mode.

What types of objects are used by your rules?

Depending upon the types of objects acted upon by your rules, the execution mode you choose differs.

Working memory objects or ruleset parameters

If the objects you use in your rules are passed with ruleset parameters, we recommend you use the sequential or Fastpath execution modes. If they are objects that you inserted into a working memory, we recommend you use the RetePlus or Fastpath execution modes.

Bindings

Bindings are heterogeneous when rules do not operate on the same classes. When bindings are heterogeneous, the rules may have condition parts of different heights. See Table 2.

Table 2 Heterogeneous bindings

Rule1	... when{A();B()} ...
Rule2	... when{A()} ...
Rule3	... when{B()} ...

If your rules define heterogeneous bindings, we recommend you use the RetePlus or Fastpath execution modes.

Bindings are homogeneous when all the rules operate on the same class (the same kind and number of objects), but introduce different tests. See Table 3.

Table 3 Homogeneous bindings

Rule1	... when{Person(age == 12);} ...
Rule2	... when{Person(age > 20);} ...

If your rules define homogeneous bindings, we recommend you use the sequential execution mode.

What is the impact of rule actions?

Depending upon the types of effects generated by your rules on execution, the execution mode you choose differs.

Modifications to the working memory

If the actions of your rules manipulate working memory objects, with the use of the IRL keywords insert, retract, or update, then you must use the RetePlus execution mode. Because these keywords entail modifications to the working memory, the rule engine reevaluates subsequent rules.

If you use another execution mode, the rule engine will not reevaluate subsequent rules after the modifications.

Rule chaining

When your rule actions trigger modifications in the working memory or the parameters, and when they do pattern matching on objects that have no relationship, like people and hobbies, there is chaining between your rules.

For example:

```
SILVER LEVEL CUSTOMER, GREATER THAN $5000 purchase
promote to GOLD LEVEL
GOLD LEVEL CUSTOMER, GREATER THAN $5000 purchase
apply 25% discount
```

You can see there is chaining between these two rules because they do pattern matching on two different objects customer and purchase and that the second one will modify the level attribute of the customer object.

Basically, if you know your rule actions will trigger the execution of other rules, then you should use the RetePlus execution mode.

What sort of tests do you find in rule conditions?

The execution mode will differ depending on what type of conditions you use in your rules.

Tests that require a working memory

If you have rule conditions that test the existence of an object or gather items in a collection directly in working memory, with the IRL keywords exists or collect, without in or from constructs, then we recommend you use the RetePlus or Fastpath execution modes.

Regular pattern for tests

If the tests in rule conditions have the same pattern and order, such as the tests that are generated from decision tables, we recommend you use the Fastpath execution mode.

If the order of tests in rule conditions is not regular, then use the RetePlus or sequential execution modes.

What priorities have you set on your rules?

If you have set static priorities on your rules, you can use any algorithm. However, if you set dynamic priorities, that is, priorities that are defined as an expression, then you must use the RetePlus execution mode.

Summary

You can use Table 4 on page 25 as a reference in making your decision when you choose an execution mode for a rule task.

Table 4 Choosing an execution mode

In your rule task:	RetePlus	Sequential	Fastpath
Compliance and validation application	X	✓	✓
Computation application	✓	X	X
Correlation application	✓	X	✓
Stateful application	✓	X	X
Working memory objects	✓	✓	✓
Rule chaining	✓	X	X
Tests on existence or collection items directly in working memory	✓	✓	✓
Shared test patterns	X	X	✓
Heterogeneous bindings	✓	X	✓
Dynamic priorities	✓	X	X
Runtime rule selection that selects a few rules among many	✓	✓	X
Numerous rules	X	✓	✓

Do you have tips and tricks for business rule applications?

For many people, business rule software is a new technology, and they do not yet have an innate sense of the performance profile of an application that uses a rule engine. For example, is invoking a set of 1,000 rules faster or slower than a database call? Of course, the answer depends on a variety of circumstances. Here are some best practices when dealing with performance issues in business rule applications.

For applications with stringent performance targets or service level agreements, you should consider continuous performance monitoring. By investing daily in a small amount of performance monitoring, you can avoid expensive refactoring due to performance issues found at the end of your development cycle. This is particularly important for business rule applications where the relationship between the rules being authored and the code being invoked at runtime is not as obvious as with a traditional single-programming-language procedural software system.

Consider the basic sequence of actions required to invoke a ruleset using JRules:

1. Ruleset archive is parsed, creating an IlrRuleset
2. Instantiation of an IlrContext for the IlrRuleset
3. Creation of input data (IN,INOUT parameters) for the ruleset
4. Optionally add objects to Working Memory
5. Execute the ruleset
6. Retrieve INOUT and OUT parameters
7. Optionally retrieve objects from Working Memory
8. Reset IlrContext for subsequent execution

Operation 1 is costly, and for a large ruleset containing more than 10000 rules, it can take several minutes. However, it is a one-time cost, as the resulting IlrRuleset should be pooled for subsequent reuse. Rule Execution Server provides out-of-the-box caching of IlrRuleset and IlrContext.

Operation 3 is often slow, as this is where your code hits various expensive backend systems, (such as databases) to retrieve the data required by the rule engine. Caching within your code might be a useful optimization here.

The time required for Operation 5 depends on the details of the rules, ruleflows, and rule tasks within your ruleset. The rules typically invoke methods in your Java Executable Object Model (XOM). It is important to determine the amount of time spent within your code versus the time spent within the rule engine or within the rules.

The documentation of the product contains some useful information around Optimizing Execution.

Do you have a Rule Engine Performance cost breakdown?

The performance cost for a JRules application may looks something like Figure 15.

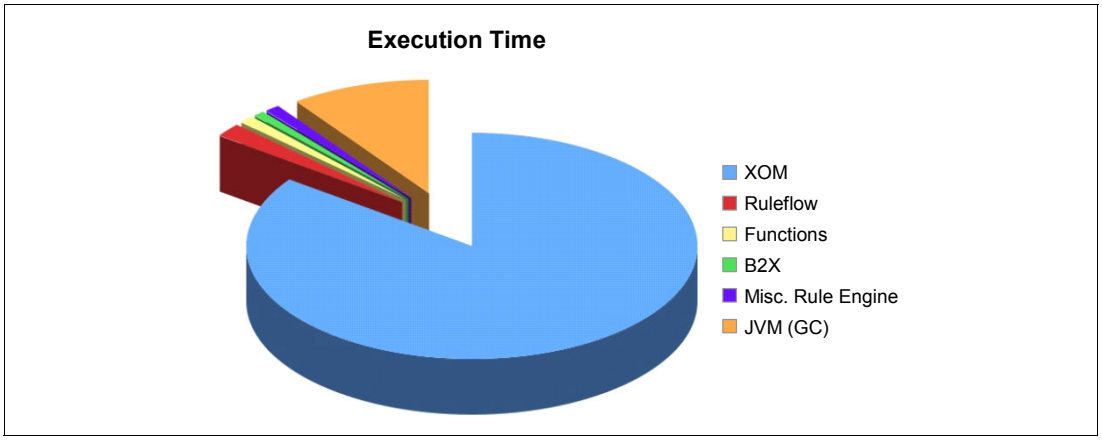

Figure 15 Execution Time

The vast majority of application time is typically spent within the XOM code invoked by the rules. It is therefore critical to get a good understanding of which methods within your XOM code are going to be invoked by rules and roughly how frequently. This may also include synchronization considerations, because if you have 10 rule engines concurrently calling a synchronized method within your XOM, your throughput is going to suffer.

Enhancing the performance of Rule Studio

Rule Studio is the development environment for business rule applications. It is integrated into Eclipse. Developers can take advantage of this integration to develop their Java projects with rule projects. Developers and architects can therefore use Rule Studio to integrate and deploy the business rule application into their enterprise client application. Architects, developers, and business analysts use Rule Studio to design the business rule application, author, review, and debug business rules. Rule Studio also has tools for keeping the rules synchronized with the Rule Team Server repository.

Can Rule Studio build time have an impact on the productivity of the development?

The time it takes to build large rule projects can become a drain on productivity, and it is something we have worked hard on improving in JRules 7.0. The language team in particular is responsible for converting the business-level rule metaphors, like Action Rules and Decision Tables, to executable IRL code. They have made some great improvements in JRules 7.0, and everyone has been anxious to quantify them using real customer projects.

We were able to perform detailed build performance analysis on six rule projects and one Java XOM project from a major international bank. All projects were rebuilt (”clean all” inside Rule Studio) and their build times measured. Note that full builds like this are not typically required during rule authoring, as we incrementally compile rule artifacts.

Aggregated artifact statistics for the six projects:

- Ruleflows: 300
- Variable Sets: 5
- Decision Tables: 363
- BOM classes: 1,860
- Vocabulary elements: 1,876
- Action Rules: 4,206
- Functions: 2
- IRL Rules: 2

As you can see in Figure 16, we achieved a two-fold performance increase almost across the board. Note that Rule Analysis in JRules 7.0 performs more semantic checking than Rule Analysis in JRules 6.7, so it is not an apples-to-apples comparison.

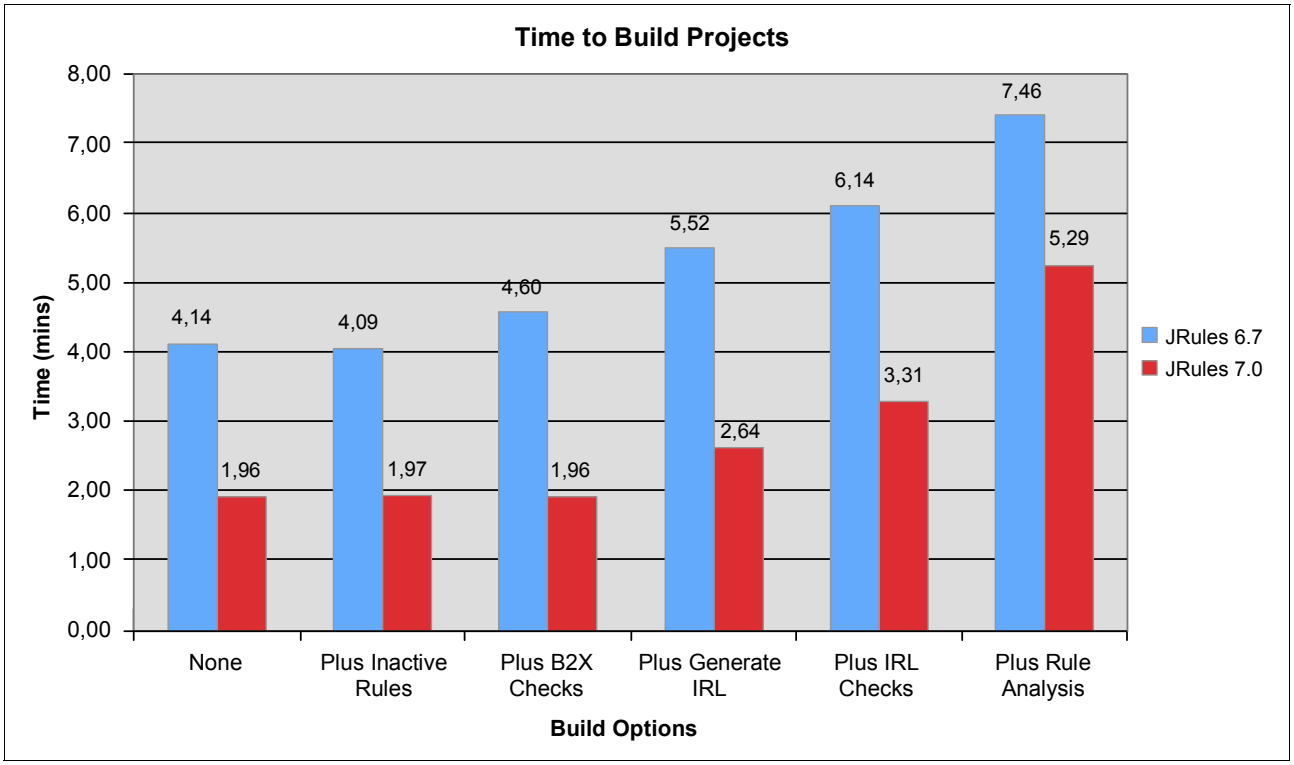

Figure 16 Build Time

All the tests were run on a development machine on a development machine running Java 5 with a 1 GB heap size. The results are good and they should make you more productive, but work continues to build performance, as it impacts your workflow and experience with the product.

Do you have recommendations on how to bench an application?

Recommendations for avoiding performance issues in your rule projects include:

- Understand, document, and communicate your performance objectives.
- Consider putting continuous performance measurement in place to provide you with a safety net as you modify your XOM and rules. It is much easier to see the impact of adding a synchronized method call to the XOM, or adding a rule with a complex join if you are receiving performance or throughput reports every morning.
- Performance test with realistic data. There is no point optimizing for a dummy data set that bears no resemblance to what you will see in production.
- Understand the frequency with which the methods in your XOM are invoked by rules, particularly if you are doing lots of inferencing using the RETE algorithm.
- Spend time optimizing your XOM where profitable.
- Spend time getting to know the features of your profiler.

Conclusion

This paper is intended to help you improve the performance of IBM WebSphere ILOG JRules 7.0. There are a number of different ways to configure the BRMS modules, which might affect performance. In this paper, the focus for improving performance was to increase the speed of execution, and to scale the application to meet the service level agreement. You can, more often than not, achieve a performance improvement by checking the default settings and assessing whether you can change parameters to get the best mix of speed and versatility for your application. This paper provides details on which parameters to check and what to change them to, depending on the type of your application.

With every new release of the BRMS suite comes new features and new possibilities to affect performance. With this in mind, ensure that you have an up-to-date version of this paper so that you make use of the performance improvements of these features.

The Author who wrote this IBM Redpaper

Pierre-André Paumelle is Performance Architect, IBM ILOG Business Rule Management Systems, with over 15 years of experience designing and developing enterprise applications with C++ and Java (J2EE). He worked side by side with customers for eight years as a consultant on ILOG products, especially those in the BRMS product line. For the past five years, he has worked on the Enterprise Integration (BRMS) team as an architect and team lead. This team is in charge of Rules Execution Server, Decision Warehouse and Decision Validation Services.

Contributors

Christophe Borde is a Product Marketing Manager for IBM WebSphere ILOG Business Rule Management product line. Before joining the business rule team more than four years ago, Christophe worked as Product Manager for the ILOG visualization product line for over two years. Previously, he worked for ILOG UK as Technical Manager and Business Development Manager for Northern Europe. Before this, Christophe held various positions within ILOG from technical support engineer to pre-sales consultant and with Thomson CSF as software engineer. Christophe earned a Master of Science in Computer Science and a Specialized Master in Marketing Management.

Albin Carpentier is a software architect on the IBM ILOG Business Rule Management Systems, with over six years of experience developing Java enterprise applications. He worked on the Enterprise Integration team as a developer in charge of management models and consoles. For the past two years, he has worked on the Rule Team Server team as a software architect, and participates actively in the integration of Decision Validation Services.

George Harley is a senior IBM developer with over a decade of experience designing and developing enterprise applications with C++, Java, XML and model-driven technologies. George ran the Rule Execution Server benchmarks on zLinux.

Antoine Melki is a software architect with over a decade of experience developing and designing various parts of ILOG Business Rule Management System. He has been the main architect and team lead for ILOG BRMS Rule Team Server since its creation, four years ago. Rule Team Server is a highly scalable rule repository exposed through a web-based rule management environment for business users.

Daniel Selman is a software architect a team lead for IBM WebSphere ILOG JRules Rule Studio a rule development environment for technical users based on the Eclipse platform. Daniel has worked as an architect and product manager on a variety of Java C C++ and J2EE applications. Daniel blogs at**:**

https://www.ibm.com/developerworks/mydeveloperworks/blogs/brms/?lang=en

Antony Viaud is a Product Manager for WebSphere Business Rules Management System, responsible for rule development requirements, focusing on Rule Studio and rule engine modules. Prior to this position, he was a Technical Account Manager with ILOG US, specialized in Business Rules Management System for financial services,

Mark Wilkinson is ILOG Collateral Manager, IBM Software Group. His responsibilities include the management of print collateral relating to the IBM ILOG products. Before joining IBM in 2009, Mark worked as the marketing communications writer and newsletter editor at ILOG S.A. for more than 11 years.

Yee Pin Yheng is a Product Manager for WebSphere Business Rules Management System for the last three years, responsible for enterprise integration requirements, focusing on Rule Execution Server functionalities, integration stacks with Java EE application servers and Rule Team Server architecture. Prior to the position, he was a staff applications developer in ILOG. He led the development effort of JRules integration for various BPM products including WebSphere Process Server, MQ Workflow and BEA WebLogic Integration.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document REDP-4632-00 was created or updated on January 20, 2010.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks